# 第7章 图

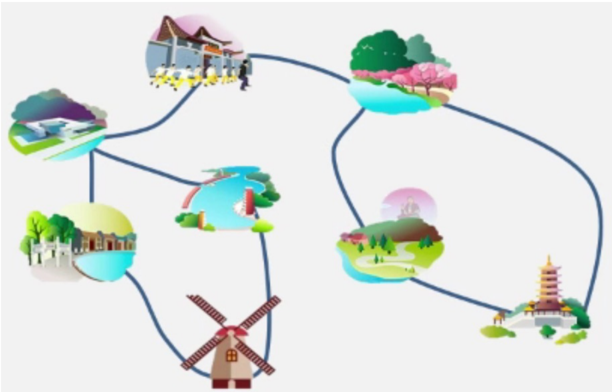
杭州電子科技大学
HANGZHOU DIANZI UNIVERSITY

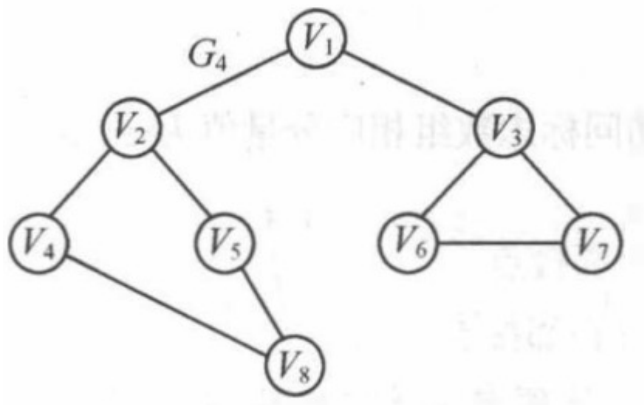
# 7.3 图的遍历

杭州電子科技大学 HANGZHOU DIANZI UNIVERSITY

遍历定义：

从已给的连通图中某一顶点出发，沿着一些边访遍图中所有的顶点，且使每个顶点仅被访问一次，就叫做图的遍历，它是图的基本运算。






遍历实质：找每个顶点的邻接点的过程。

# 7.3 图的遍历



图的特点：

图中可能存在回路，且图的任一顶点都可能与其他顶点相通，在访问完某个顶点之后可能会沿着某些边又回到曾经访问过的顶点。

怎样避免重复访问？

解决思路：设置辅助数组visited[n]，用来标记每个被访问过的顶点。

- 初始状态visited[i]为0
- 顶点i被访问，改visited[i]为1，防止被多次访问

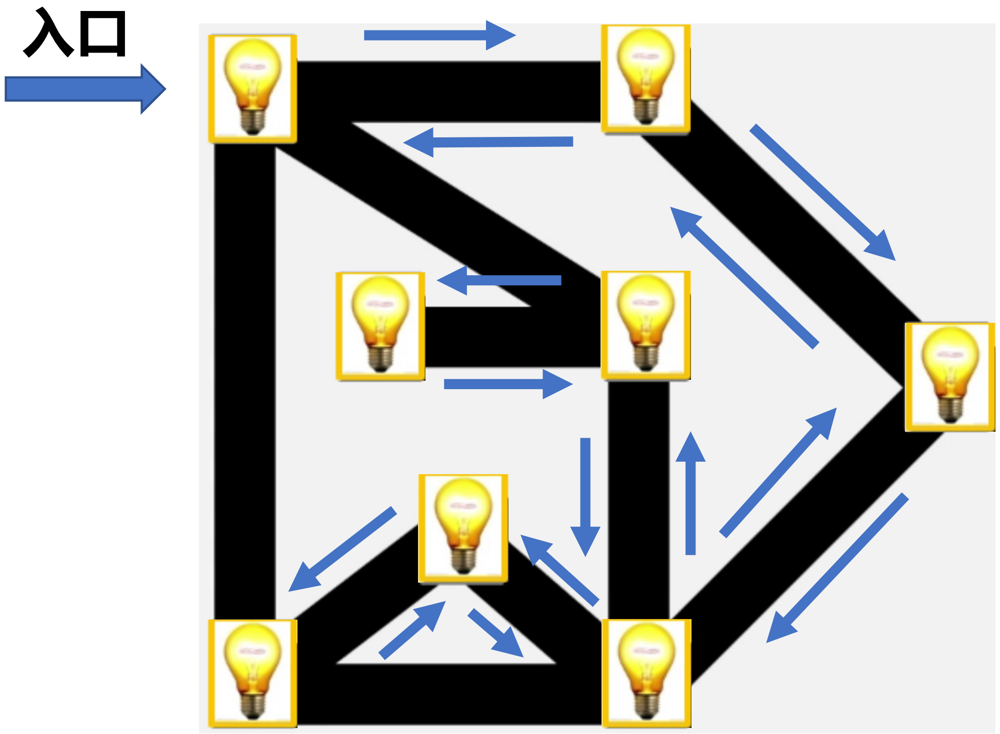
# 7.3 图的遍历



图常用的遍历：

- 深度优先搜索（Depth First Search——DFS）
- 广度优先搜索（Breadth First Search——BFS）

# 深度优先遍历（DFS）




点亮迷宫中所有的灯

# 深度优先遍历（DFS）

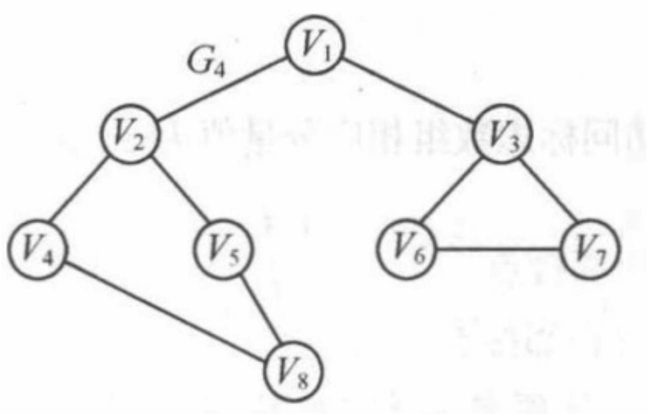
方法:

- 在访问图中的某一起始顶点$v$后，由$v$出发，访问它的任一邻接顶点$w_1$；
- 再从$w_1$出发，访问与$w_1$邻接但还未被访问过的顶点$w_2$；
- 然后再从$w_2$出发，进行类似的访问，…
- 如此进行下去，直至到达所有的邻接顶点都被访问过的顶点$u$为止。
- 接着，退回一步，退到前一次刚访问过的顶点，看是否还有其它没有被访问的邻接顶点。
  - 如果有，则访问此顶点，之后再从此顶点出发，进行与前述类似的访问;
  - 如果没有，就再退回一步进行搜索。重复上述过程，直到连通图中所有顶点都被访问过为止。

# 深度优先遍历（DFS）



例:




顶点访问次序:

$v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_8 \rightarrow v_5 \rightarrow v_3 \rightarrow v_6 \rightarrow v_7$

$v_1 \rightarrow v_2 \rightarrow v_5 \rightarrow v_8 \rightarrow v_4 \rightarrow v_3 \rightarrow v_6 \rightarrow v_7$

$v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_8 \rightarrow v_5 \rightarrow v_3 \rightarrow v_7 \rightarrow v_6$

$v_1 \rightarrow v_2 \rightarrow v_5 \rightarrow v_8 \rightarrow v_4 \rightarrow v_3 \rightarrow v_7 \rightarrow v_6$

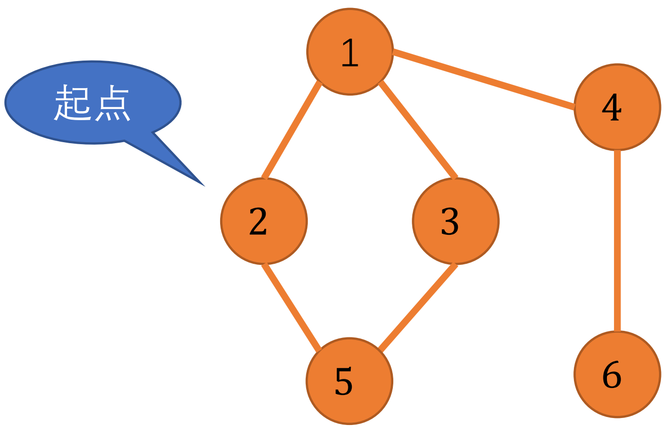
$v_1 \rightarrow v_3 \rightarrow v_6 \rightarrow v_7 \rightarrow v_2 \rightarrow v_4 \rightarrow v_8 \rightarrow v_5$

……

连通图的深度优先遍历类似与树的先根遍历

# 2. 深度优先搜索遍历算法的实现

邻接矩阵表示的无向图深度遍历实现：




邻接矩阵

$$
\begin{array}{c}
 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6
\end{array}
\begin{array}{c}
\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
\begin{bmatrix}
0 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0
\end{bmatrix}
\end{array}
$$

辅助数组visited[n]

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| visited[i] | 1 | 1 | 1 | 1 | 1 | 1 |

DFS结果

2 ➔ 1 ➔ 3 ➔ 5 ➔ 4 ➔ 6

# 2. 深度优先搜索遍历算法的实现

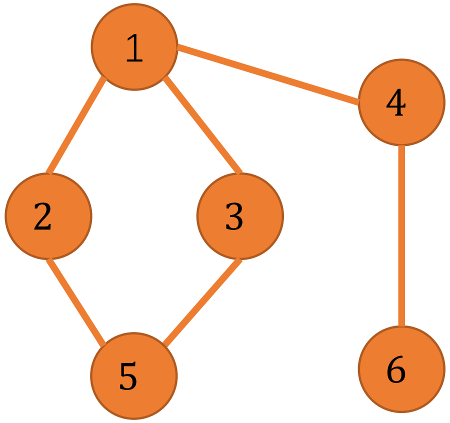
杭州電子科技大学 HANGZHOU DIANZI UNIVERSITY

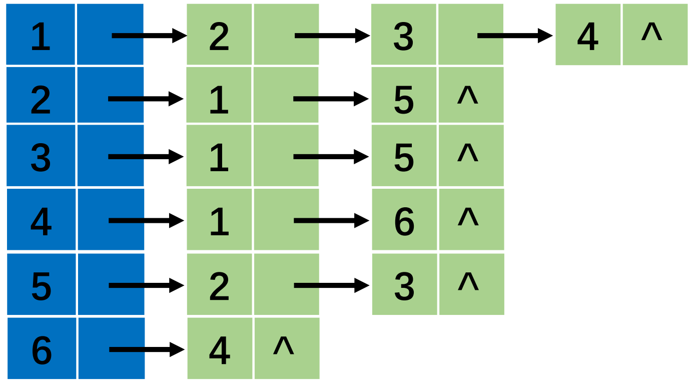
算法6.5 采用邻接矩阵表示图的深度优先搜索遍历

```
void DFS(AMGraph G, int v){                    //图G为邻接矩阵类型
    cout<<v;   visited[v] = true;              //访问第v个顶点
    for(w=0; w<G.vexnum; w++){                 //依次检查邻接矩阵v所在的行
        if((G.arcs[v][w]!=0) && (!visited[w]))
            DFS(G, w);
    //w是v的邻接点，如果w未访问，则递归调用DFS
}
```

# 2. 深度优先搜索遍历算法的实现

邻接表表示的无向图深度遍历实现：



# DFS算法效率分析

- 用邻接矩阵来表示图，遍历图中的每一个顶点都要从头扫描该顶点所在行，时间复杂度为$O(n^2)$。
- 用邻接表来表示图，访问所有单链表上的表结点即可完成遍历，加上访问n个头结点的时间，时间复杂度为$O(n+e)$。

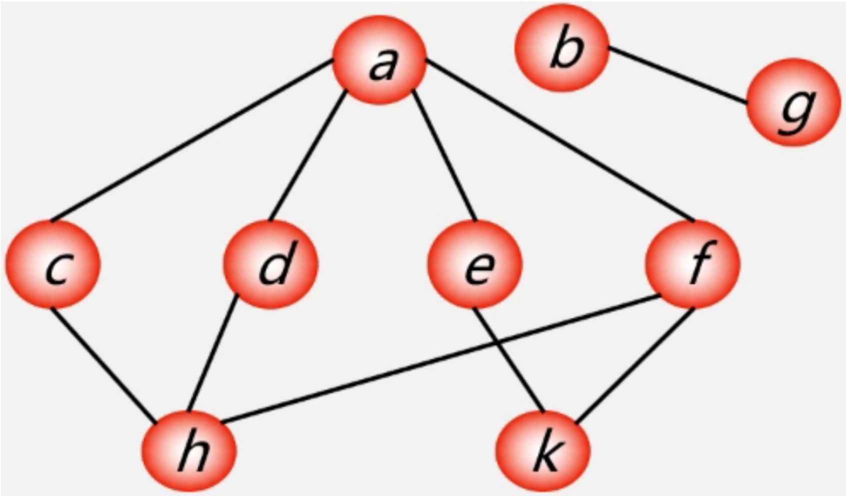
**结论：**

稠密图适于在邻接矩阵上进行深度遍历；

稀疏图适于在邻接表上进行深度遍历。

# 非连通图的遍历



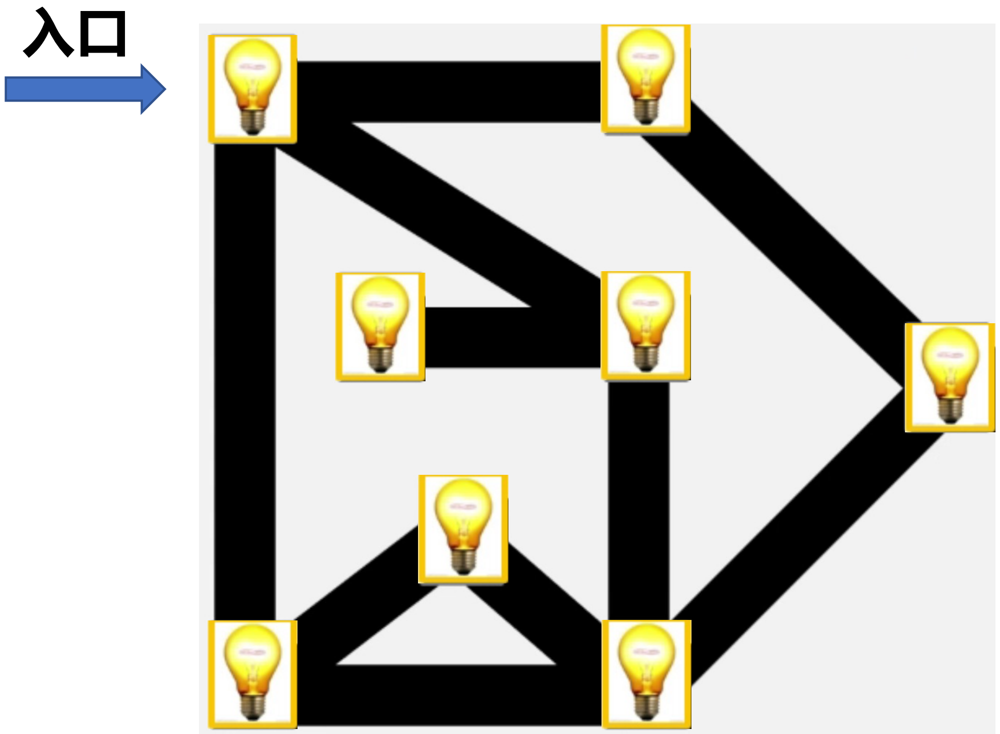



顶点访问次序：

a c h d f k e b g

# 广度优先遍历（BFS）



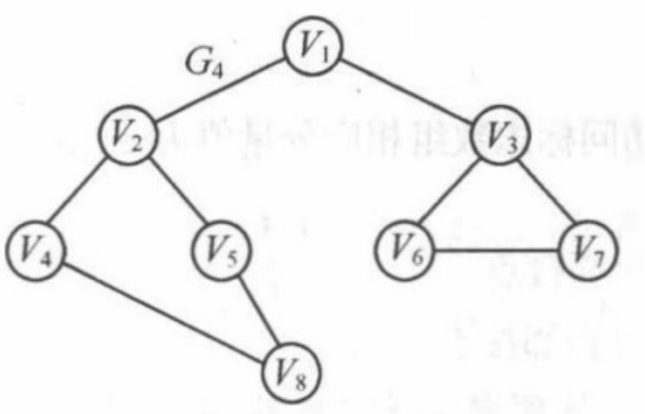



点亮迷宫中所有的灯

# 广度优先遍历（BFS）

**方法**：

- 从图的某一结点出发，首先依次访问该结点的所有邻接结点 $v_{i_1}, v_{i_2}, \ldots, v_{i_n}$，再按这些顶点被访问的先后次序依次访问与它们相邻接的所有未被访问的顶点；
- 重复此过程，直至所有顶点均被访问为止。

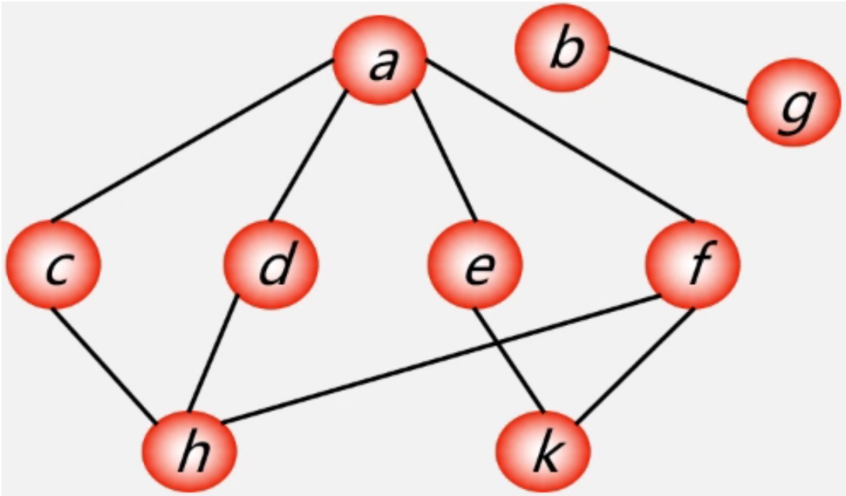



广度优先遍历：

$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow v_7 \rightarrow v_8$

# 非连通图的广度遍历

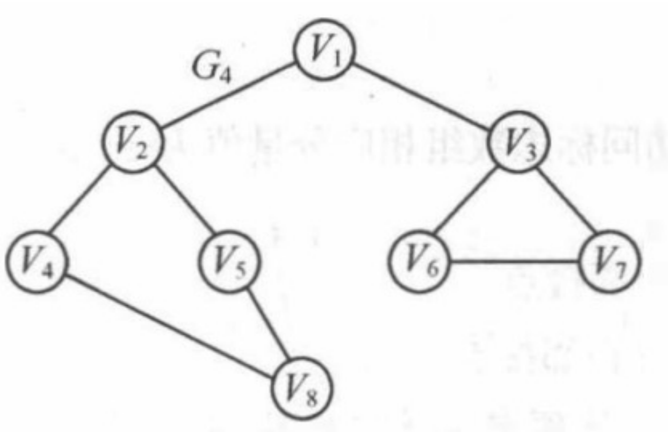


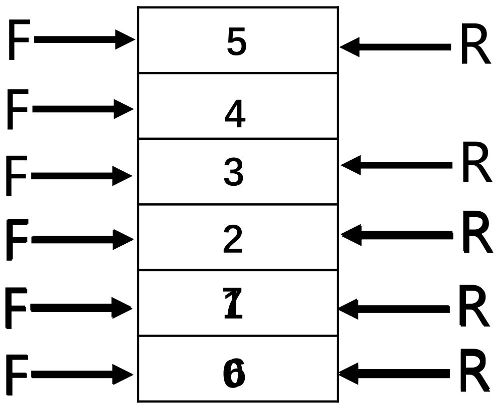


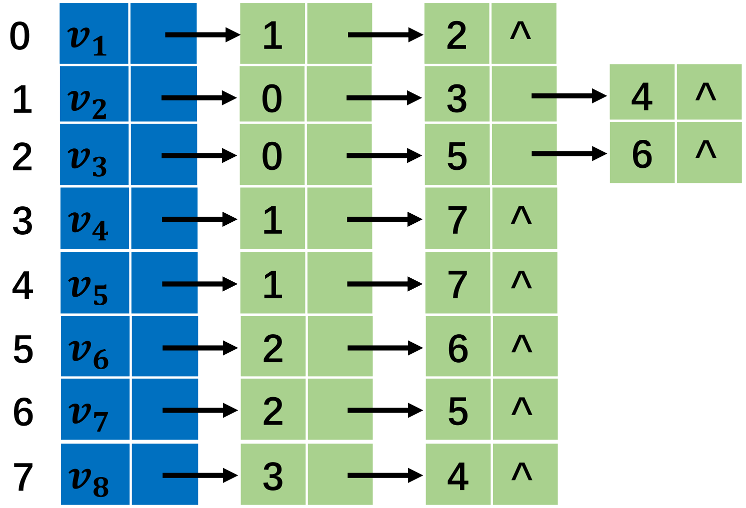

顶点访问次序：

a c d e f h k b g

实现:




| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| visited[i] | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow v_7 \rightarrow v_8$

# 广度优先遍历

算法6.7 按广度优先非递归遍历连通图G

```
void BFS(Graph G, int v){                                    //按广度优先非递归遍历连通图G
        cout<<v; visited[v] = true;                          //访问第v个顶点
        InitQueue(Q);                                        //辅助队列Q初始化，置空
        EnQueue(Q, v);                                       //v进队
        while(!QueueEmpty(Q)){                               //队列非空
              DeQueue(Q, u);                                 //队头元素出队并置为u
              for(w=FirstAdjVex(G, u); w>=0; w=NextAdjVex(G, u, w))
                   if(!visited[w]){                          //w为u的尚未访问的邻接结点
                        cout<<w; visited[w]=true;  EnQueue(Q, w);    //w进队
                   }
         }
}
```

# BFS算法效率分析

- 如果使用邻接矩阵，则BFS对于每一个被访问到的顶点，都要循环检测矩阵中的整整一行（n个元素），总的时间代价为$O(n^2)$。
- 用邻接表来表示图，访问所有单链表上的表结点即可完成遍历，加上访问n个头结点的时间，时间复杂度为$O(n+e)$。
- DFS和BFS的时间复杂只与存储结构（邻接矩阵或邻接表）有关，而与搜索路径无关。

# 第7章 图

# 7.4 图的应用

**最小生成树**

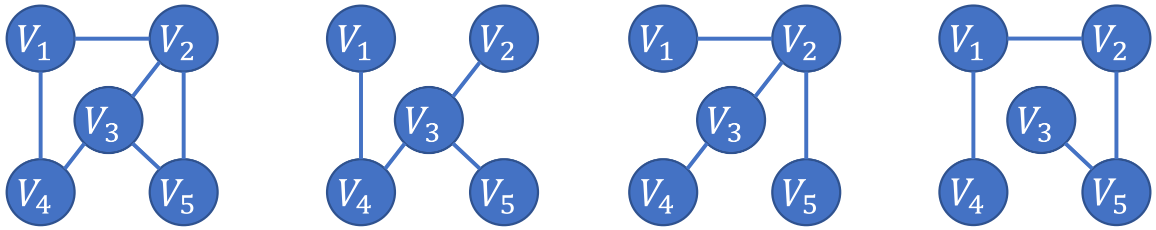
最短路径

拓扑排序

关键路径

# 概念回顾——生成树

生成树：所有顶点均由边连接在一起，但不存在回路的图




一个图可以有许多棵不同的生成树

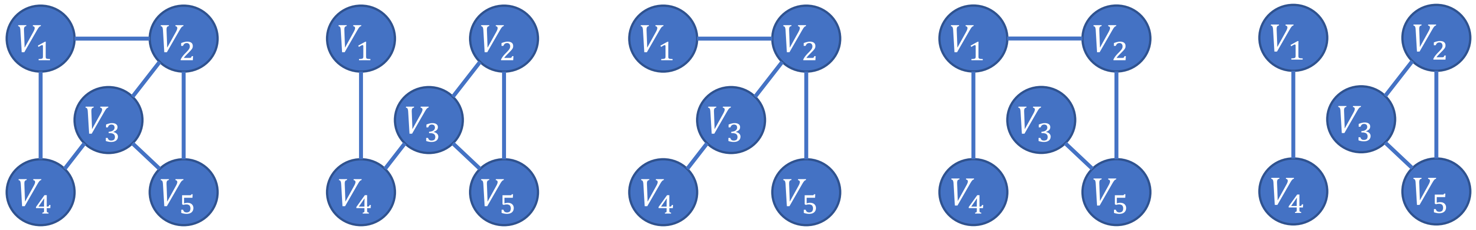
所有生成树有具有以下共同点：

- 生成树的顶点个数与图的顶点个数相同；
- 生成树是图的极小连通子图，去掉一条边则非连通；
- 一个有n个顶点的连通图的生成树有n-1条边；
- 在生成树中再添加一条必然形成回路；
- 生成树中任意两个顶点间的路径是唯一的；

# 概念回顾——生成树



生成树：所有顶点均由边连接在一起，但不存在回路的图

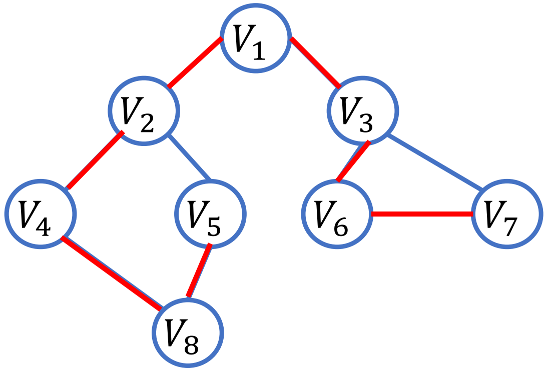



一个图可以有许多棵不同的生成树

所有生成树有具有一下共同点：

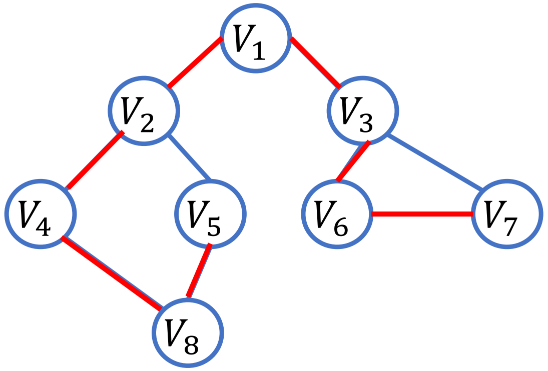
含n个顶点n-1条边的图不一定是生成树。

# 无向图的生成树

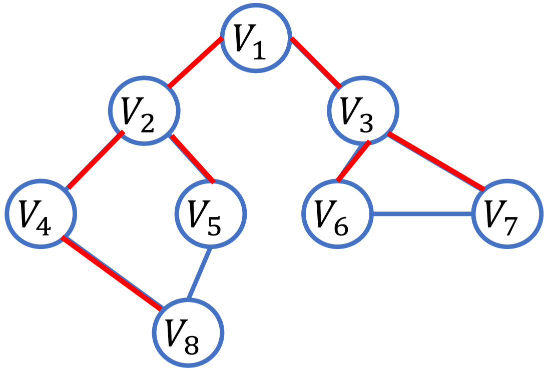



深度优先生成树

# 无向图的生成树



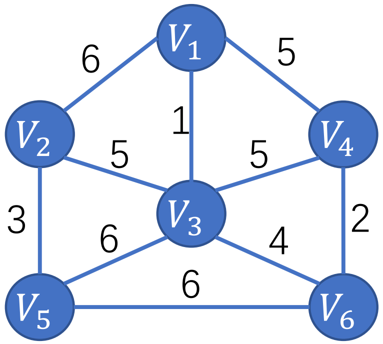

深度优先生成树

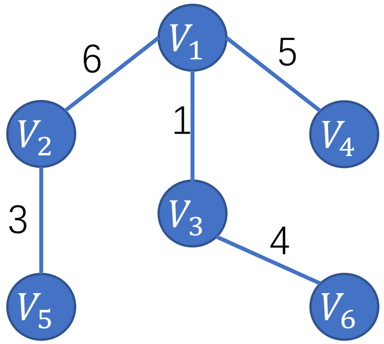



广度优先生成树

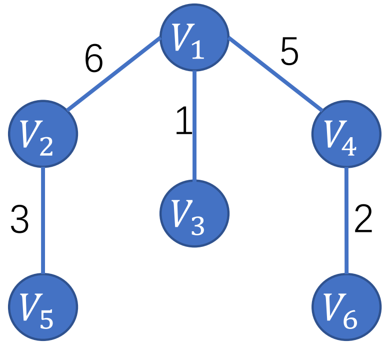
设图G=(V,E)是个连通图，当从图任一顶点出发遍历图G时，将边集E(G)分成两个集合T(G)和B(G)。其中**T(G)**是遍历图时所经过的边的集合，B(G)是遍历图时未经过的边的集合。显然，G1(V,T)是图G的极小连通子图。即子图G1是连通图G的生成树。

# 最小生成树




各边权值和=19




各边权值和=17

最小生成树：给定一个无向网络，在该网络的所有生成树中，使得各边权值之和最小的那棵树称为该网的最小生成树，也叫**最小代价生成树**。

# 最小生成树的典型用途

杭州电子科技大学 HANGZHOU DIANZI UNIVERSITY

- 欲在n个城市间建立通信网，则n个城市应铺n-1条线路
- 但因为每条线路都会有对应的经济成本，而n个城市最多有n(n-1)/2条线路，那么，如何选择n-1条线路，使总费用最少?

数学模型：

顶点———表示城市，有n个；

边————表示线路，有n-1条；

边的权值—表示线路的经济代价；

连通网——表示n个城市间通信网。

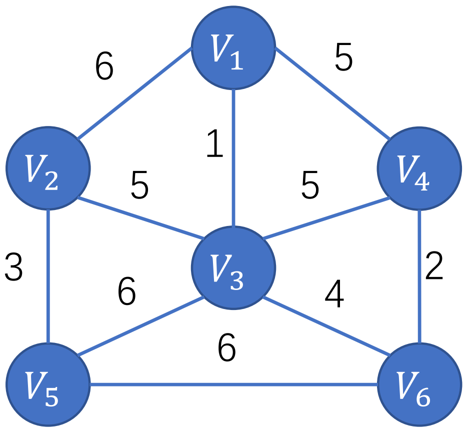
显然此连通网是一个生成树!

# 构造最小生成树 Minimum Cost Spanning Tree

杭州電子科技大学 HANGZHOU DIANZI UNIVERSITY

构造最小生成树的算法很多，其中多数算法都利用了MST的性质。

MST性质：设N=(V，E)是一个连通网，U是顶点集V的一个非空子集。若边(u,v)是一条具有最小权值的边，其中u属于U，v属于V-U，则必存在一棵包含边(u,v)的最小生成树。




N=(V, {E})

V={v1, v2, v3, v4, v5, v6}

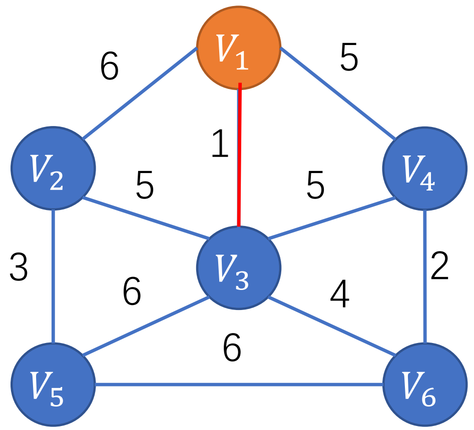
E={(v1, v2), (v1, v3), (v1, v4), (v2, v3), (v2, v5), (v3, v4), (v3, v5), (v3, v6), (v4, v6), (v5, v6), }

# 构造最小生成树 **M**inimum **S**panning **T**ree

杭州电子科技大学 HANGZHOU DIANZI UNIVERSITY

构造最小生成树的算法很多，其中多数算法都利用了MST的性质。

MST性质：设N=(V，E)是一个连通网，U是顶点集V的一个非空子集。若边(u,v)是一条具有最小权值的边，其中u属于U，v属于V-U，则必存在一棵包含边(u,v)的最小生成树




N=(V, {E})

V={v1, v2, v3, v4, v5, v6}

E={(v1, v2), (v1, v3), (v1, v4), (v2, v3), (v2, v5), (v3, v4), (v3, v5), (v3, v6), (v4, v6), (v5, v6), }

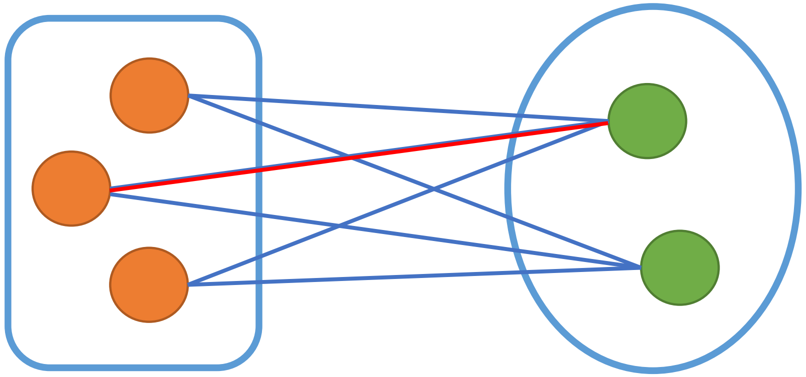
U={v1}　　　V-U={v2, v3, v4, v5, v6}

# MST性质解释

在生成树的构造过程中，图中n个顶点分属两个集合：

- 已落在生成树上的顶点集：U
- 尚未落在生成树上的顶点集：V-U

接下来则应在所有连通U中顶点和V-U中顶点的边中选取**权值最小的边**



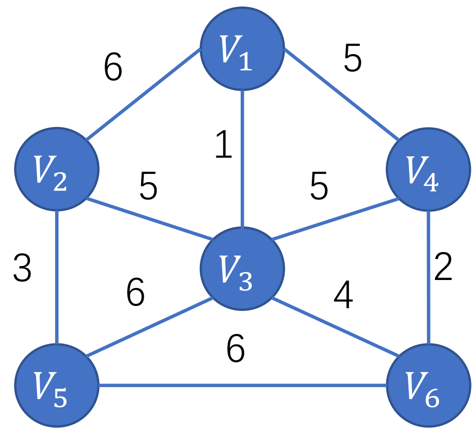
生成树上的顶点集合U　　尚未落在生成树上的顶点集合(V-U)

# 构造最小生成树方法一：普里姆(Prim)算法

杭州電子科技大學 HANGZHOU DIANZI UNIVERSITY

## 算法思想

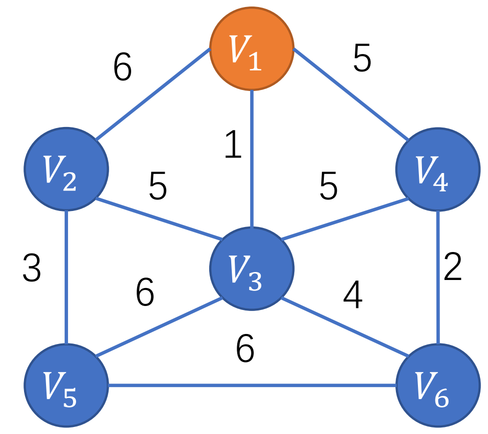
- 设N=(V, {E})时连通网，TE是N上最小生成树中边的集合
- 初始令U={u0}, (u0∈V), TE={}。



# 构造最小生成树方法一：普里姆(Prim)算法

## 算法思想

- 设N=(V, {E})时连通网，TE是N上最小生成树中边的集合
- 初始令U={u0}, (u0∈V), TE={}。
- 在所有u ∈ U，v ∈V-U的边(u,v) ∈E中，找一条代价最小的边(u0,v0)

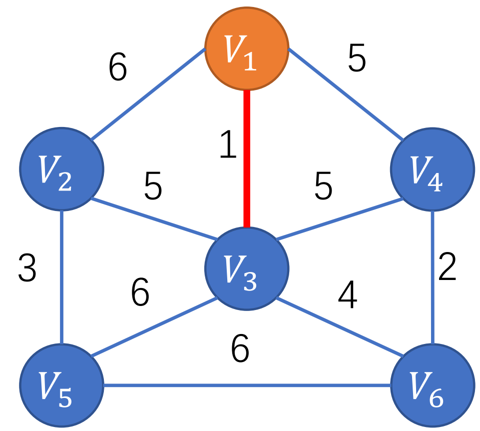


# 构造最小生成树方法一：普里姆(Prim)算法

杭州電子科技大学 HANGZHOU DIANZI UNIVERSITY

## 算法思想

- 设N=(V, {E})时连通网，TE是N上最小生成树中边的集合
- 初始令U={u0}, (u0∈V), TE={}。
- 在所有u ∈ U，v ∈V-U的边(u,v) ∈E中，找一条代价最小的边(u0,v0)
- 将(u0,v0)并入集合TE，同时v0并入U

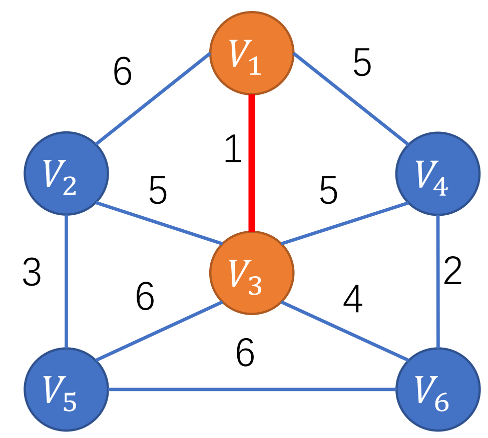


# 构造最小生成树方法一：普里姆(Prim)算法

杭州電子科技大学 HANGZHOU DIANZI UNIVERSITY

## 算法思想

- 设N=(V, {E})时连通网，TE是N上最小生成树中边的集合
- 初始令U={u0}, (u0∈V), TE={}。
- 在所有u ∈ U，v ∈V-U的边(u,v) ∈E中，找一条代价最小的边(u0,v0)
- 将(u0,v0)并入集合TE，同时v0并入U
- 重复上述操作至U=V为止，则T=(V, {TE})为N的最小生成树

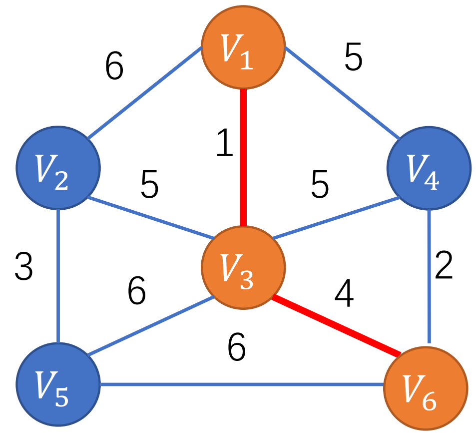


# 构造最小生成树方法一：普里姆(Prim)算法

杭州電子科技大學 HANGZHOU DIANZI UNIVERSITY

## 算法思想

- 设N=(V, {E})时连通网，TE是N上最小生成树中边的集合
- 初始令U={u0}, (u0∈V), TE={}。
- 在所有u ∈ U，v ∈V-U的边(u,v) ∈E中，找一条代价最小的边(u0,v0)
- 将(u0,v0)并入集合TE，同时v0并入U
- 重复上述操作至U=V为止，则T=(V, {TE})为N的最小生成树

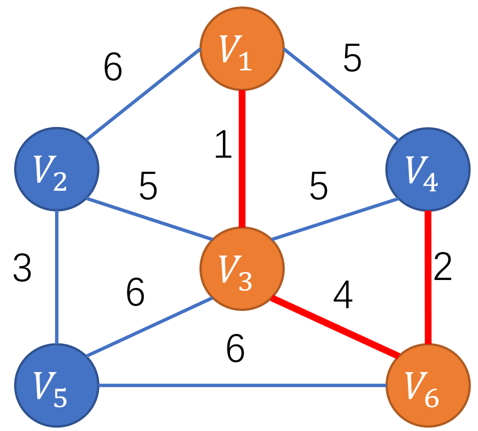


# 构造最小生成树方法一：普里姆(Prim)算法



## 算法思想

- 设N=(V, {E})时连通网，TE是N上最小生成树中边的集合
- 初始令U={u0}, (u0∈V), TE={}。
- 在所有u ∈ U，v ∈V-U的边(u,v) ∈E中，找一条代价最小的边(u0,v0)
- 将(u0,v0)并入集合TE，同时v0并入U
- 重复上述操作至U=V为止，则T=(V, {TE})为N的最小生成树

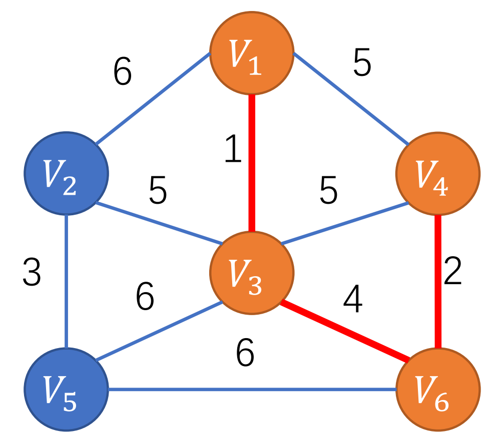


# 构造最小生成树方法一：普里姆(Prim)算法

杭州电子科技大学 HANGZHOU DIANZI UNIVERSITY

## 算法思想

- 设N=(V, {E})时连通网，TE是N上最小生成树中边的集合
- 初始令U={u0}, (u0∈V), TE={}。
- 在所有u ∈ U，v ∈V-U的边(u,v) ∈E中，找一条代价最小的边(u0,v0)
- 将(u0,v0)并入集合TE，同时v0并入U
- 重复上述操作至U=V为止，则T=(V, {TE})为N的最小生成树

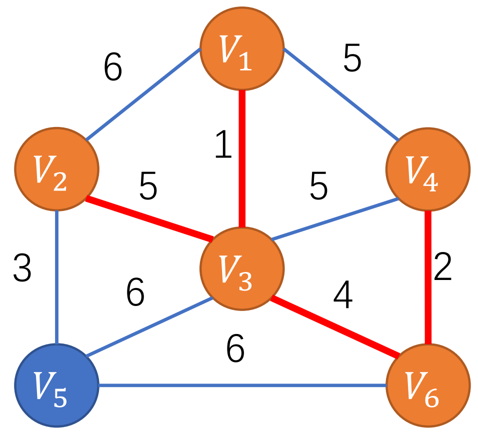


# 构造最小生成树方法一：普里姆(Prim)算法

杭州電子科技大學 HANGZHOU DIANZI UNIVERSITY

## 算法思想

- 设N=(V, {E})时连通网，TE是N上最小生成树中边的集合
- 初始令U={u0}, (u0∈V), TE={}。
- 在所有u ∈ U，v ∈V-U的边(u,v) ∈E中，找一条代价最小的边(u0,v0)
- 将(u0,v0)并入集合TE，同时v0并入U
- 重复上述操作至U=V为止，则T=(V, {TE})为N的最小生成树

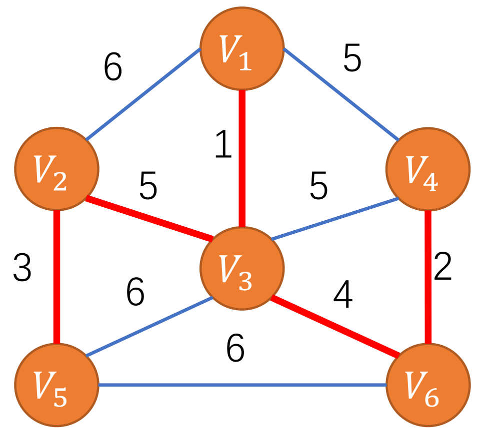


# 构造最小生成树方法一：普里姆(Prim)算法



## 算法思想

- 设N=(V, {E})时连通网，TE是N上最小生成树中边的集合
- 初始令U={u0}, (u0∈V), TE={}。
- 在所有u ∈ U，v ∈V-U的边(u,v) ∈E中，找一条代价最小的边(u0,v0)
- 将(u0,v0)并入集合TE，同时v0并入U
- 重复上述操作至U=V为止，则T=(V, {TE})为N的最小生成树

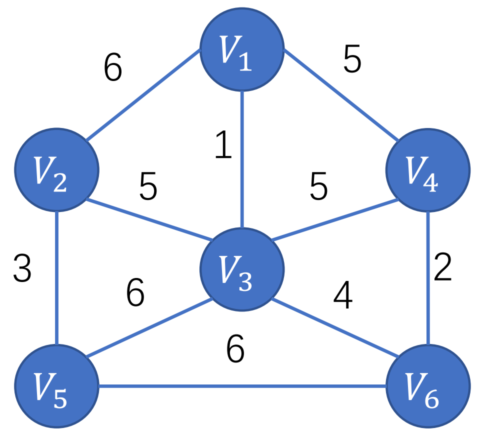


# 构造最小生成树方法一：克鲁斯卡尔(Kruskal)算法



## 算法思想

- 设连通网N=(V,E)，令最小生成树初始状态为只有n个顶点而无边的非连通图T=(V, {})，每个顶点自成一个连通分量

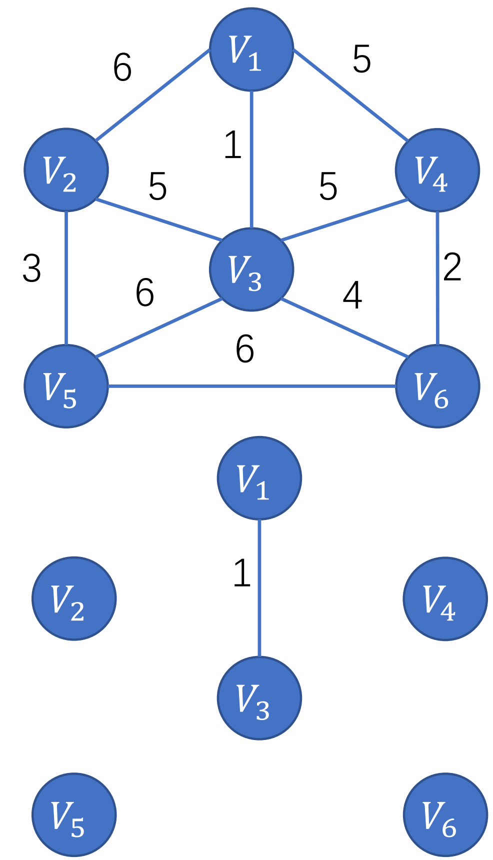


# 构造最小生成树方法一：克鲁斯卡尔(Kruskal)算法



## 算法思想

- 设连通网N=(V,E)，令最小生成树初始状态为只有n个顶点而无边的非连通图T=(V, {})，每个顶点自成一个连通分量
- 在 E 中选取代价最小的边，若该边依附的顶点落在T中不同的连通分量上(即:不能形成环)，则将此边加入到T中；否则，舍去此边，选取下一条代价最小的边。

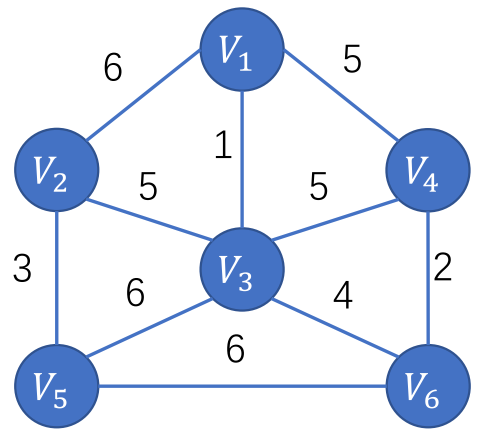


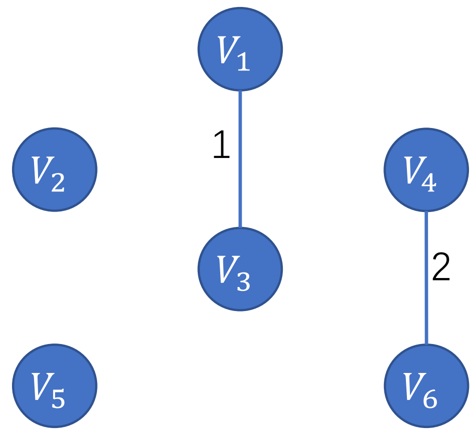
# 构造最小生成树方法一：克鲁斯卡尔(Kruskal)算法



## 算法思想

- 设连通网N=(V,E)，令最小生成树初始状态为只有n个顶点而无边的非连通图T=(V, {})，每个顶点自成一个连通分量
- 在 E 中选取代价最小的边，若该边依附的顶点落在T中不同的连通分量上(即:不能形成环)，则将此边加入到T中；否则，舍去此边，选取下一条代价最小的边。
- 依此类推，直至 T 中所有顶点都在同一连通分量上为止



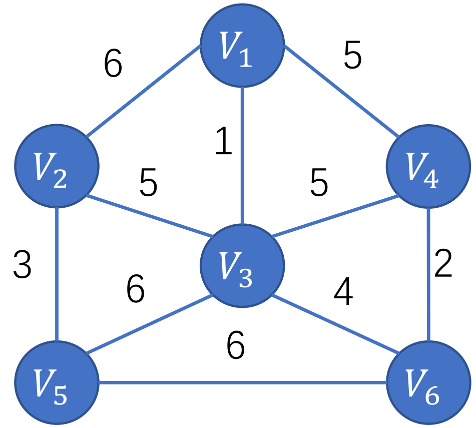



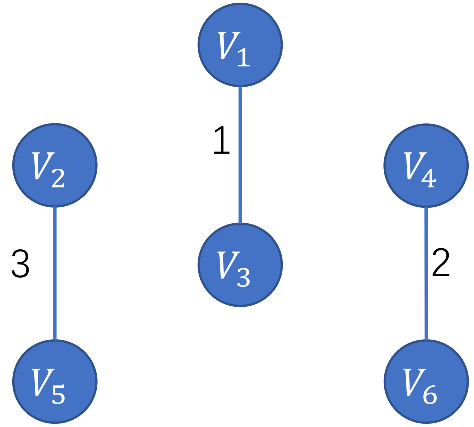
# 构造最小生成树方法一：克鲁斯卡尔(Kruskal)算法



## 算法思想

- 设连通网N=(V,E)，令最小生成树初始状态为只有n个顶点而无边的非连通图T=(V, {})，每个顶点自成一个连通分量
- 在 E 中选取代价最小的边，若该边依附的顶点落在T中不同的连通分量上(即:不能形成环)，则将此边加入到T中；否则，舍去此边，选取下一条代价最小的边。
- 依此类推，直至 T 中所有顶点都在同一连通分量上为止



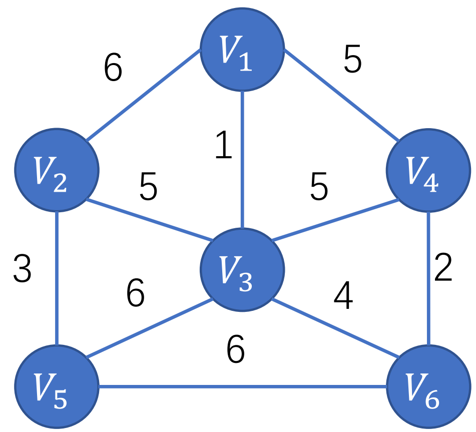



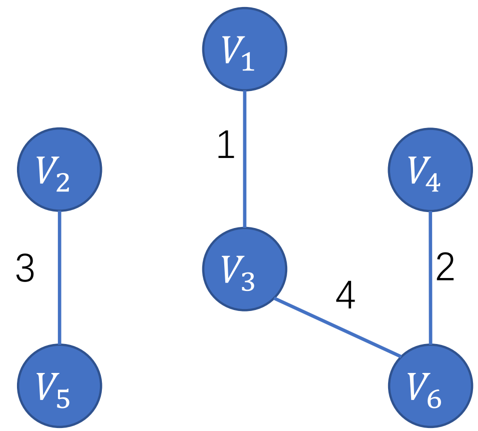
# 构造最小生成树方法一：克鲁斯卡尔(Kruskal)算法



## 算法思想

- 设连通网N=(V,E)，令最小生成树初始状态为只有n个顶点而无边的非连通图T=(V, {})，每个顶点自成一个连通分量
- 在 E 中选取代价最小的边，若该边依附的顶点落在T中不同的连通分量上(即:不能形成环)，则将此边加入到T中；否则，舍去此边，选取下一条代价最小的边。
- 依此类推，直至 T 中所有顶点都在同一连通分量上为止



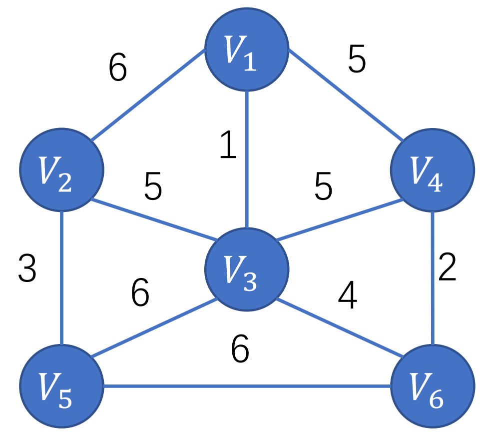



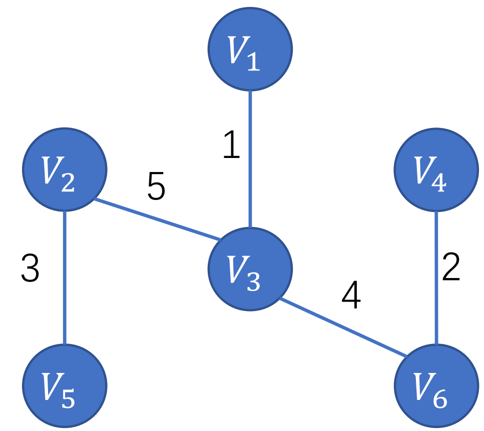
# 构造最小生成树方法一：克鲁斯卡尔(Kruskal)算法

杭州電子科技大學 HANGZHOU DIANZI UNIVERSITY

## 算法思想

- 设连通网N=(V,E)，令最小生成树初始状态为只有n个顶点而无边的非连通图T=(V, {})，每个顶点自成一个连通分量
- 在 E 中选取代价最小的边，若该边依附的顶点落在T中不同的连通分量上(即:不能形成环)，则将此边加入到T中；否则，舍去此边，选取下一条代价最小的边。
- 依此类推，直至 T 中所有顶点都在同一连通分量上为止



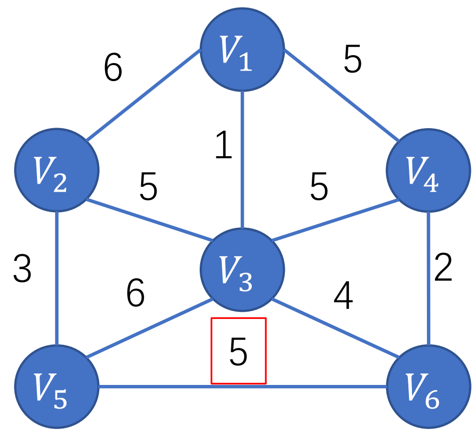



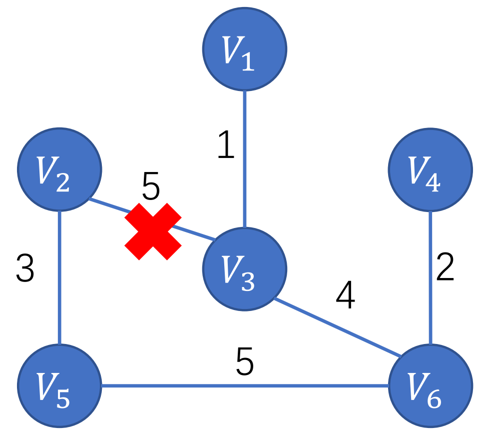
# 构造最小生成树方法一：克鲁斯卡尔(Kruskal)算法

## 算法思想

- 设连通网N=(V,E)，令最小生成树初始状态为只有n个顶点而无边的非连通图T=(V, {})，每个顶点自成一个连通分量
- 在 E 中选取代价最小的边，若该边依附的顶点落在T中不同的连通分量上(即:不能形成环)，则将 T中；否则，舍去此边，选取下一条代价最
- 依此类推，直至 T 中所有顶点都在同一连通分量上为止

最小生成树
可能不唯一



## 两种算法比较

| 算法名 | 普里姆算法 | 克鲁斯卡尔算法 |
| --- | --- | --- |
| 算法思想 | 选择点 | 选择边 |
| 时间复杂度 | $O(n^2)$(n为顶点数) | O(eloge)(e为边数) |
| 适应范围 | 稠密图 | 稀疏图 |

# 7.4 图的应用

最小生成树

**最短路径**

拓扑排序

关键路径

# 最短路径



典型用途：交通网络的问题——从甲地到乙地之间是否有公路连通？在有多条通路的情况下，哪一条路最短？

# 最短路彳

典型用

通路的

通？在有多条

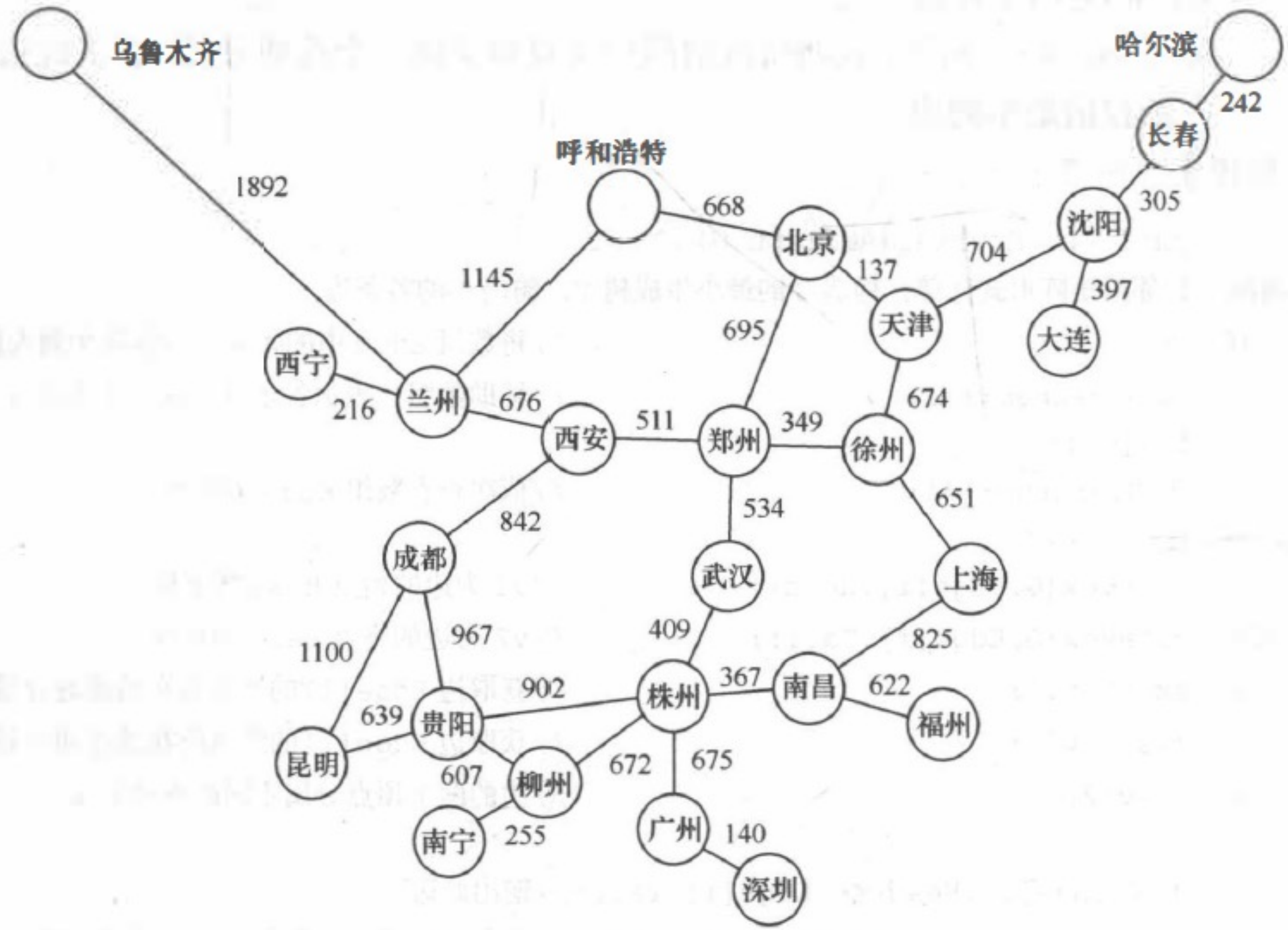



图 6.21　一个表示交通网的例图

# 最短路径



典型用途：交通网络的问题——从甲地到乙地之间是否有公路连通？在有多条通路的情况下，哪一条路最短？

交通网络用有向网来表示：

顶点——表示地点，

弧——表示两个地点有路连通，

弧上的权值——表示两地点之间的距离、交通费或途中花费的时间等

如何能够使一个地点到另一个地点的运输时间最短或运费最省？这就是一个求两个地点间的最短路径问题。

# 最短路径

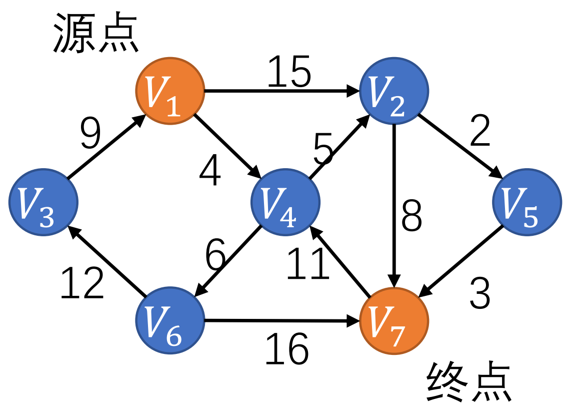
杭州電子科技大学 HANGZHOU DIANZI UNIVERSITY

问题抽象：在有向网中A点（源点）到达B点（终点）的多条路径中，寻找一条各边权值之和最小的路径，即最短路径。

最短路径与最小生成树不同，路径上不一定包含n个顶点，也不一定包含n-1条边

# 最短路径

## 第一种情况：两点间最短路径

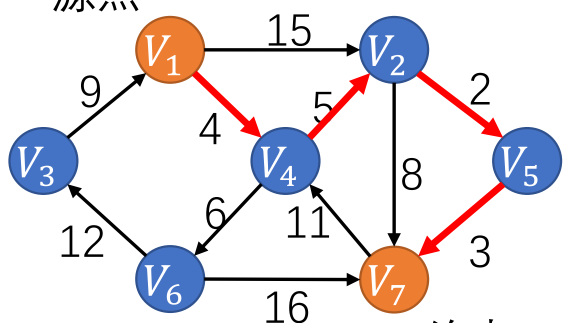



从 v1 到 v7 的路径及路径长度：

| | |
|---|---|
| v1、v2、v5、v7： | 20 |
| v1、v4、v2、v5、v7： | 14 |
| v1、v2、v7： | 23 |
| v1、v4、v2、v7： | 17 |
| v1、v4、v6、v7： | 26 |

# 最短路径

第一种情况：两点间最短路径




从 v1 到 v7 的路径及路径长度：

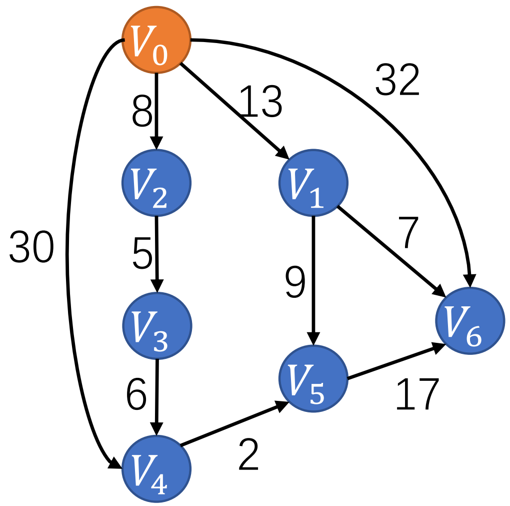
| | |
|---|---|
| v1、v2、v5、v7： | 20 |
| v1、v4、v2、v5、v7： | 14 |
| v1、v2、v7： | 23 |
| v1、v4、v2、v7： | 17 |
| v1、v4、v6、v7： | 26 |

扩展：

求图中任意两个顶点间的最短路径

# 最短路径

## 第二种情况：某源点到其他各点最短路径




| V0到各顶点最短路径 | 路径长度 |
|---|---|
| 到v1：(v0, v1) | 13 |
| 到v2：(v0, v2) | 8 |
| 到v3：(v0, v2, v3) | 13 |
| 到v4：(v0, v2, v3, v4) | 19 |
| 到v5：(v0, v2, v3, v4, v5) | 21 |
| 到v6：(v0, v1, v6) | 20 |

# 最短路径

两种常见的最短路径问题：

一、单源最短路径——用Dijkstra（迪杰斯特拉）算法

二、所有顶点间的最短路径——用Floyd（弗洛伊德）算法

# 艾兹格·W·迪科斯彻 (Edsger Wybe Dijkstra)

杭州电子科技大学 HANGZHOU DIANZI UNIVERSITY

他是几位影响力最大的计算科学的奠基人之一，也是少数同时从工程和理论的角度塑造这个新学科的人。他的根本性贡献覆盖了很多领域，包括：编译器、操作系统、分布式系统、程序设计、编程语言、程序验证、软件工程、图论等等。他的很多论文为后人开拓了整个新的研究领域。我们现在熟悉的一些标准概念，比如互斥、死锁、信号量等，都是Dijkstra发明和定义的。1994年时有人对约1000名计算机科学家进行了问卷调查，选出了38篇这个领域最有影响力的论文，其中有五篇是Dijkstra写的。

“有效的程序员不应该浪费很多时间用于程序调试，他们应该一开始就不要把故障引入”

# Dijistra算法

杭州電子科技大學
HANGZHOU DIANZI UNIVERSITY

1.初始化：先找出从源点v0到各终点vk的直达路径(v0,vk)，

即通过一条弧达到的路径。

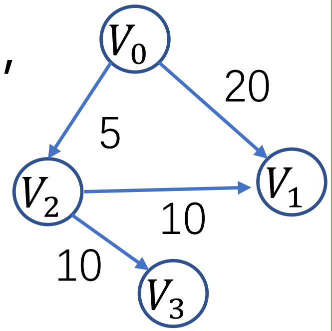
2.选择：从这些路径中找出一条长度最短的路径(v0,u)。

3.更新：然后对其余各条路径进行适当调整：

若在图中存在弧(u,vk)，且(v0,u)+(u,vk)<(v0,vk), 则

以路径(v0,u,vk)代替(v0,vk)

在调整后的各条路径中，再找长度最短的路径，依此类推



# Dijistra算法



**迪杰斯特拉算法按路径长度递增次序产生最短路径**

1、把V分成两组：

(1) S：已求出最短路径的顶点的集合。

(2) T=V-S：尚未确定最短路径的顶点集合。

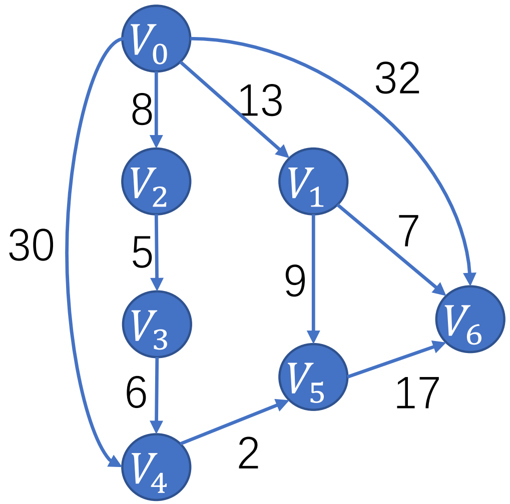
2、将T中顶点按最短路径递增的次序加入到S中，

保证：(1) 从源点v0到S中各顶点的最短路径长度都不大于从v0到T中任何顶点的最短路径长度

(2) 每个顶点对应一个距离值：

S中顶点：从v0到此顶点的最短路径长度

T中顶点：从v0到此顶点的只包括S中顶点作中间顶点的最短路径长度

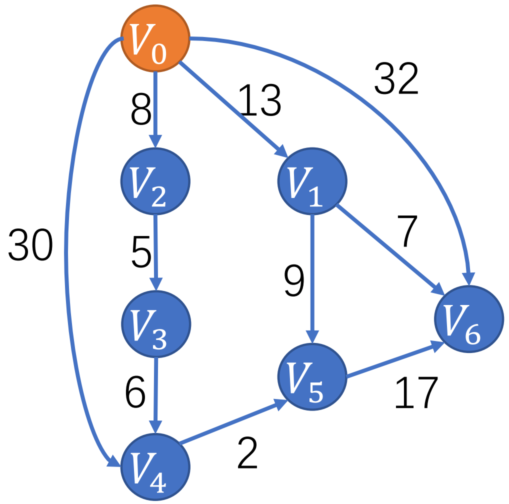


# Dijistra算法步骤：

**初始时令S={v0}，T={其余顶点}。**

T中顶点对应的距离值用辅助数组D存放。

D[i]初值：若<v0,vi>存在，则其为权值，否则为∞。




**S={ v0 }**　　D[j]=min{D[i]| vi ∈T}

| 终点 | 从v0到各终点的最短路径及长度 | | | | | |
|---|---|---|---|---|---|---|
| | i=1 | i=2 | i=3 | i=4 | i=5 | i=6 |
| v1 | | | | | | |
| v2 | | | | | | |
| v3 | | | | | | |
| v4 | | | | | | |
| v5 | | | | | | |
| v6 | | | | | | |
| vj | | | | | | |
| 距离 | | | | | | |
| 路径 | | | | | | |

# Dijistra算法步骤：

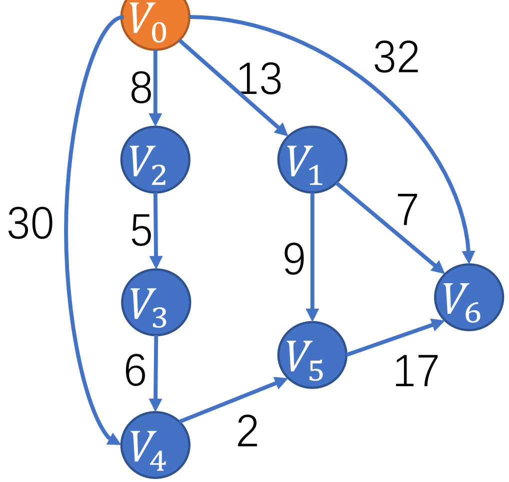


**初始时令S={v0}，T={其余顶点}。**

T中顶点对应的距离值用辅助数组D存放。

D[i]初值：若<v0,vi>存在，则其为权值，否则为∞。

从T中选取一个距离值最小的顶点vj，加入S。




S={ v0 }

D[j]=min{D[i]|vi ∈T}

| 终点 | 从v0到各终点的最短路径及长度 | | | | | |
|---|---|---|---|---|---|---|
| | i=1 | i=2 | i=3 | i=4 | i=5 | i=6 |
| v1 | 13 | | | | | |
| v2 | 8 | | | | | |
| v3 | ∞ | | | | | |
| v4 | 30 | | | | | |
| v5 | ∞ | | | | | |
| v6 | 32 | | | | | |
| vj | | | | | | |
| 距离 | | | | | | |
| 路径 | | | | | | |

# Dijistra算法步骤：

**初始时令S={v0}，T={其余顶点}。**

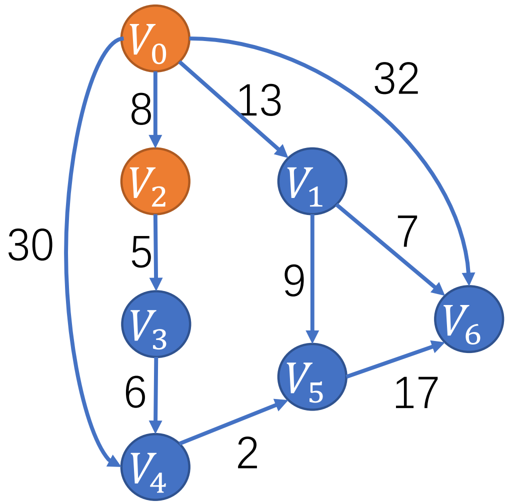
T中顶点对应的距离值用辅助数组D存放。

D[i]初值：若<v0,vi>存在，则其为权值，否则为∞。

从T中选取一个距离值最小的顶点vj，加入S。

对T中顶点的距离值进行修改：

若加进vj作中间顶点，从v0到v的距离值比不加vj的路径要短，则修改此距离值。




**S={ v0, v2 }**

D[j]=min{D[i]|vi ∈T}

| 终点 | 从v0到各终点的最短路径及长度 | | | | | |
|---|---|---|---|---|---|---|
| | i=1 | i=2 | i=3 | i=4 | i=5 | i=6 |
| v1 | 13 | 13 | | | | |
| v2 | 8 | - | | | | |
| v3 | ∞ | 13 | | | | |
| v4 | 30 | 30 | | | | |
| v5 | ∞ | ∞ | | | | |
| v6 | 32 | 32 | | | | |
| vj | v2 | | | | | |
| 距离 | 8 | | | | | |
| 路径 | (v0,v2) | | | | | |

# Dijistra算法步骤：



**初始时令S={v0}，T={其余顶点}。**

T中顶点对应的距离值用辅助数组D存放。

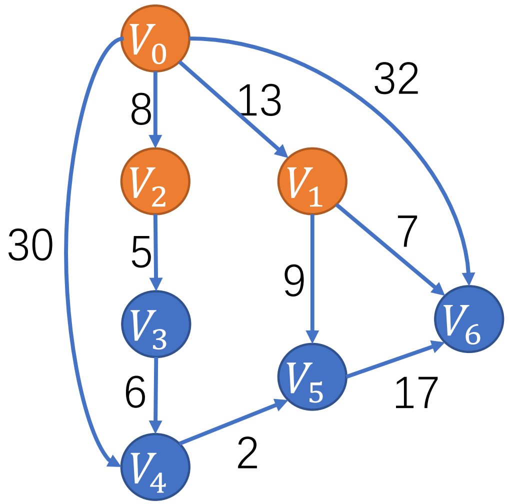
D[i]初值：若<v0,vi>存在，则其为权值，否则为∞。

从T中选取一个距离值最小的顶点vj，加入S。

对T中顶点的距离值进行修改：

若加进vj作中间顶点，从v0到v的距离值比不加vj的路径要短，则修改此距离值。

重复上述步骤，直到S=V为止。




S={ v0, v2, v1 }

D[j]=min{D[i]|vi ∈T}

| 终点 | 从v0到各终点的最短路径及长度 | | | | | |
|---|---|---|---|---|---|---|
| | i=1 | i=2 | i=3 | i=4 | i=5 | i=6 |
| v1 | 13 | 13 | - | - | - | - |
| v2 | 8 | - | - | - | - | - |
| v3 | ∞ | 13 | 13 | | | |
| v4 | 30 | 30 | 30 | | | |
| v5 | ∞ | ∞ | 22 | | | |
| v6 | 32 | 32 | 20 | | | |
| vj | v2 | v1 | | | | |
| 距离 | 8 | 13 | | | | |
| 路径 | (v0,v2) | (v0,v1) | | | | |

# Dijistra算法步骤：

**初始时令S={v0}，T={其余顶点}。**

T中顶点对应的距离值用辅助数组D存放。

D[i]初值：若<v0,vi>存在，则其为权值，否则为∞。

从T中选取一个距离值最小的顶点vj，加入S。

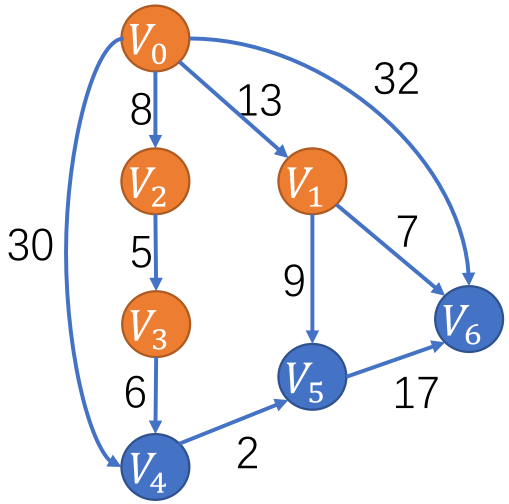
对T中顶点的距离值进行修改：

若加进vj作中间顶点，从v0到v

爹距离值比不加vj的路径要短，

则修改此距离值。

重复上述步骤，直到S=V为止。




**S={ v0, v2, v1, v3 }**

D[j]=min{D[i]|vi ∈T}

| 终点 | 从v0到各终点的最短路径及长度 | | | | | |
|---|---|---|---|---|---|---|
| | i=1 | i=2 | i=3 | i=4 | i=5 | i=6 |
| v1 | 13 | 13 | - | - | - | - |
| v2 | 8 | - | - | - | - | - |
| v3 | ∞ | 13 | 13 | - | - | - |
| v4 | 30 | 30 | 30 | 19 | | |
| v5 | ∞ | ∞ | 22 | 22 | | |
| v6 | 32 | 32 | 20 | 20 | | |
| vj | v2 | v1 | v3 | | | |
| 距离 | 8 | 13 | 8+5 | | | |
| 路径 | (v0,v2) | (v0,v1) | (v0,v2, v3) | | | |

# Dijistra算法步骤：

**初始时令S={v0}，T={其余顶点}。**

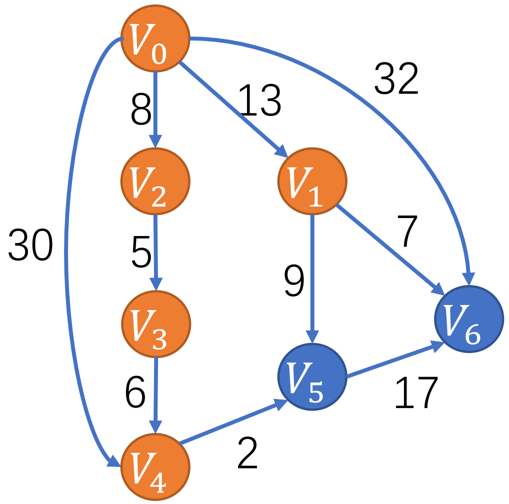
T中顶点对应的距离值用辅助数组D存放。

D[i]初值：若<v0,vi>存在，则其为权值，否则为∞。

从T中选取一个距离值最小的顶点vj，加入S。

对T中顶点的距离值进行修改：若加进vj作中间顶点，从v0到vi的距离值比不加vj的路径要短，则修改此距离值。

重复上述步骤，直到S=V为止。




S={ v0, v2, v1, v3, v4 }

D[j]=min{D[i]|vi ∈T}

| 终点 | 从v0到各终点的最短路径及长度 | | | | | |
|---|---|---|---|---|---|---|
| | i=1 | i=2 | i=3 | i=4 | i=5 | i=6 |
| v1 | 13 | 13 | - | - | - | - |
| v2 | 8 | - | - | - | - | - |
| v3 | ∞ | 13 | 13 | - | - | - |
| v4 | 30 | 30 | 30 | 19 | - | - |
| v5 | ∞ | ∞ | 22 | 22 | 21 | |
| v6 | 32 | 32 | 20 | 20 | 20 | |
| vj | v2 | v1 | v3 | v4 | | |
| 距离 | 8 | 13 | 8+5 | 8+5 +6 | | |
| 路径 | (v0,v2) | (v0,v1) | (v0,v2, v3) | (v0,v2, v3,v4) | | |

# Dijistra算法步骤：

**初始时令S={v0}，T={其余顶点}。**

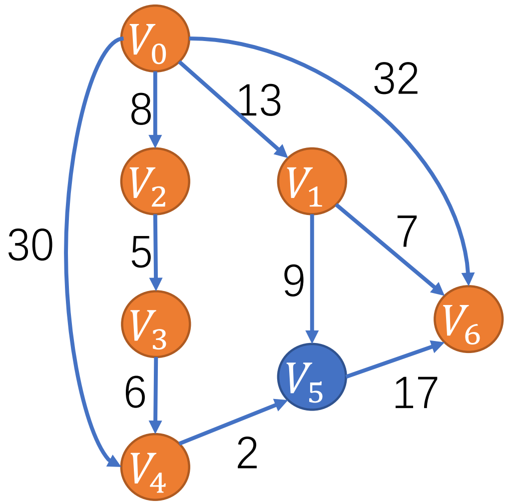
T中顶点对应的距离值用辅助数组D存放。

D[i]初值：若<v0,vi>存在，则其为权值，否则为∞。

从T中选取一个距离值最小的顶点vj，加入S。

对T中顶点的距离值进行修改：若加进vj作中间顶点，从v0到vi的距离值比不加vj的路径要短，则修改此距离值。

重复上述步骤，直到S=V为止。




**S={ v0, v2, v1, v3, v4, v6 }**

D[j]=min{D[i]|vi ∈T}

| 终点 | 从v0到各终点的最短路径及长度 | | | | | |
|---|---|---|---|---|---|---|
| | i=1 | i=2 | i=3 | i=4 | i=5 | i=6 |
| v1 | 13 | 13 | - | - | - | - |
| v2 | 8 | - | - | - | - | - |
| v3 | ∞ | 13 | 13 | - | - | - |
| v4 | 30 | 30 | 30 | 19 | - | - |
| v5 | ∞ | ∞ | 22 | 22 | 21 | 21 |
| v6 | 32 | 32 | 20 | 20 | 20 | - |
| vj | v2 | v1 | v3 | v4 | v6 | |
| 距离 | 8 | 13 | 8+5 | 8+5+6 | 13+7 | |
| 路径 | (v0,v2) | (v0,v1) | (v0,v2,v3) | (v0,v2,v3,v4) | (v0,v1,v6) | |

# Dijistra算法步骤：



**初始时令S={v0}，T={其余顶点}。**

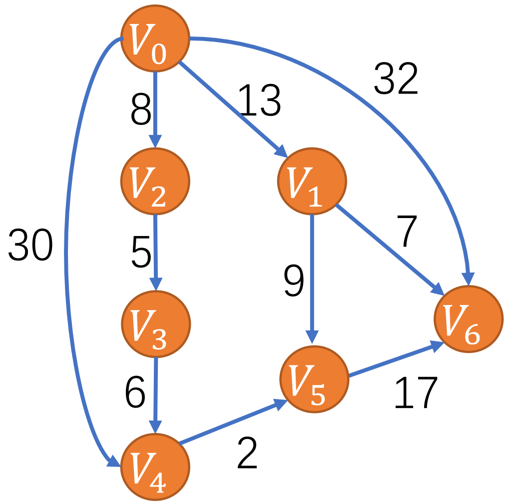
T中顶点对应的距离值用辅助数组D存放。

D[i]初值：若<v0,vi>存在，则其为权值，否则为∞。

从T中选取一个距离值最小的顶点vj，加入S。

对T中顶点的距离值进行修改：若加进vj作中间顶点，从v0到v爹距离值比不加vj的路径要短，则修改此距离值。

重复上述步骤，直到S=V为止。




S={ v0, v2, v1, v3, v4, v6, v5 } D[j]=min{D[i]|vi ∈T}

| 终点 | 从v0到各终点的最短路径及长度 | | | | | |
|---|---|---|---|---|---|---|
| | i=1 | i=2 | i=3 | i=4 | i=5 | i=6 |
| v1 | 13 | 13 | - | - | - | - |
| v2 | 8 | - | - | - | - | - |
| v3 | ∞ | 13 | 13 | - | - | - |
| v4 | 30 | 30 | 30 | 19 | - | - |
| v5 | ∞ | ∞ | 22 | 22 | 21 | 21 |
| v6 | 32 | 32 | 20 | 20 | 20 | - |
| vj | v2 | v1 | v3 | v4 | v6 | v5 |
| 距离 | 8 | 13 | 8+5 | 8+5+6 | 13+7 | 8+5+6+2 |
| 路径 | (v0,v2) | (v0,v1) | (v0,v2,v3) | (v0,v2,v3,v4) | (v0,v1,v6) | (v0,v2,v3,v4,v5) |

# 最短路径

两种常见的最短路径问题：

一、单源最短路径——用Dijkstra（迪杰斯特拉）算法

二、所有顶点间的最短路径——用Floyd（弗洛伊德）算法

# 所有顶点间的最短路径

杭州電子科技大學 HANGZHOU DIANZI UNIVERSITY

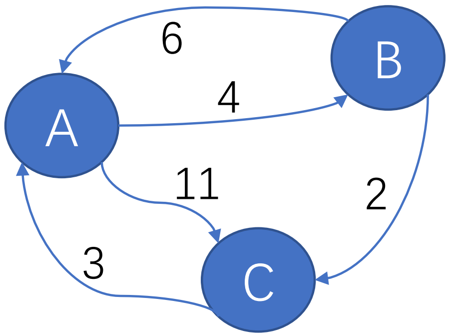
方法一：每次以一个顶点为源点，重复执行Dijkstra算法n次

方法二：佛洛伊德(Floyd)算法

算法思想：

- 逐个顶点试探
- 从vi到vj的所有可能存在的路径中选出一条长度最短的路径

# 例：采用Floyd算法，求图中个顶点之间的最短路径




初始：$\begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{pmatrix}$ 路径

| | AB | AC |
|---|---|---|
| BA | | BC |
| CA | | |

加入A：$\begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$ 路径

| | AB | AC |
|---|---|---|
| BA | | BC |
| CA | CAB | |

加入B：$\begin{pmatrix} 0 & 4 & \mathbf{6} \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$ 路径

| | AB | **ABC** |
|---|---|---|
| BA | | BC |
| CA | CAB | |

加入C：$\begin{pmatrix} 0 & 4 & \mathbf{6} \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$ 路径

| | AB | **ABC** |
|---|---|---|
| BCA | | BC |
| CA | CAB | |

求最短路径步骤：

初始时设置一个n阶方阵，令其对角线元素为0，若存在弧<vi,vj>，则对应元素为权值，否则为∞

逐步试着在原直接路径中增加中间顶点，若加入中间顶点后路径变短，则修改之；否则，维持原值。所有顶点试探完毕，算法结束。